The Geometry of Parallelism

Classical, Probabilistic, and Quantum Effects

Ugo Dal Lago
Università di Bologna, Italy & INRIA, France
ugo.dallago@unibo.it

Claudia Faggian
CNRS & Univ. Paris Diderot, France
faggian@irif.fr

Benoît Valiron
LRI, CentraleSupélec, Univ. Paris-Saclay, France
benoit.valiron@lri.fr

Akira Yoshimizu
The University of Tokyo, Japan
yoshimizu@is.s.u-tokyo.ac.jp

Abstract

We introduce a Geometry of Interaction model for higher-order quantum computation, and prove its *adequacy* for *a fully fledged quantum programming language* in which entanglement, duplication, and recursion are all available.

This model is an instance of a new framework which captures not only quantum but also classical and *probabilistic* computation. Its main feature is the ability to model *commutative effects* in a *parallel* setting. Our model comes with a multi-token machine, a proof net system, and a PCF-style language. Being based on a multi-token machine equipped with a memory, it has a concrete nature which makes it well suited for building low-level operational descriptions of higher-order languages.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]: Algebraic approaches to semantics

Keywords Geometry of Interaction, quantum computation, probabilistic computation, lambda calculus, memory structure

1. Introduction

In classical computation, information is deterministic, discrete and freely duplicable. Already in the early days [1], however, determinism has been relaxed by allowing state evolution to be probabilistic. The classical model has then been further challenged by quantum computation [2], a computation paradigm which is based on the laws of quantum mechanics.

Probabilistic and quantum computation are both justified by the very efficient algorithms they give rise to: think about Miller-Rabin primality test [3, 4], Shor's factorization [5] but also the more recent algorithms for quantum chemistry [6] or for solving linear systems of equations [7]. Finding out a way to conveniently *express* those algorithms without any reference to the underlying hardware, is then of paramount importance.

POPL'17, January 15–21, 2017, Paris, France
ACM. 978-1-4503-4660-3/17/01...$15.00
http://dx.doi.org/10.1145/3009837.3009859

This has stimulated research on programming languages for probabilistic [8, 9] and quantum computation (see [10] for a survey), and recently on higher-order functional languages [11–13]. The latter has been epitomized by variations and extensions of the λ-calculus. In order to allow compositional reasoning, it is important to give a denotational semantics to those languages; maybe surprisingly, a large body of works in this direction is closely connected to denotational and interactive models of Linear Logic [14], in the style of Game Semantics [15, 16] and the Geometry of Interaction [17].

The case of quantum computation is emblematic. The first adequate denotational model for a quantum programming language *à la* PCF, only two years old [13], marries a categorical construction for the exponentials of linear logic [18, 19] to a suitable extension of the standard model of quantum computation: the category of completely positive maps [20]. The development of an *interactive* semantics has proved to be highly nontrivial, with results which are impressive but not yet completely satisfactory. In particular, the underlying language either does not properly reflect entanglement [21–23], a key feature of quantum computation, or its expressive power is too weak, lacking recursion and duplication [24, 25]. The main reason for this difficulty lies in the inherent non-locality of entanglement [2].

We show here that Girard's Geometry of Interaction (GoI) indeed offers the right tools to deal with a fully fledged quantum programming language in which duplication and full recursion are available, when we equip GoI with an external quantum memory, a standard technique for operational models of quantum λ-calculi [11].

We go further: the approach we develop is not specific to quantum computation, and our quantum model is introduced as an instance of a new framework which models *choice effects* in a parametric way, via a *memory structure*. The memory structure comes with three operations: (1) *allocation* of fresh addresses in the memory, (2) *low-level actions* on the memory, and (3) *choice* based on the value of the memory at a given address. The notion of memory structure is flexible, the only requirement being commutativity of the operations. In Sec. 3.3 we show that different kinds of choice effects can be systematically treated: classical, probabilistic and quantum memory are all instances of this general notion. Therefore, the memory makes the model suitable to interpret classical, probabilistic and quantum functional programs. In particular, in the case of quantum memory, a low-level action is an application of unitary gate to the memory, while the choice performs a quantum measure.

The GoI model we give has a very concrete nature, as it consists of a class of token machines [26, 27]. Their distinctive feature is to be *parallel* and *multi-token* [25, 28] rather than *single-token* as in classic token machines [26]. Being multi-token means that different computational threads can interact with each other and synchronize (think of this as a multi-player game, where players are able to collaborate and exchange information). The presence of multiple tokens allows to appropriately reflect non-locality in a quantum setting, but also to generally deal with *parallelism* and *choice effects* in a satisfactory way. We discuss why this is the case when we concretely present the machine (Sec. 5).

Finally, to deal with the combination of parallelism, probabilistic side-effects and non-termination, we develop a general notion of PARS, *probabilistic abstract reduction system*. The results we establish on PARS are the key ingredient in the Adequacy proofs, but are also of independent interest. The issues at sake are non-trivial and we discuss them in the dedicated Sec. 1.2.

Contributions. We present a Geometry of Interaction (GoI) model for higher-order quantum computation, which is adequate for a quantum programming language in which entanglement, duplication, and recursion are all available. Our model comes with a multi-token machine, a proof net system, and a PCF-style language. More specifically, this paper's contributions can be summarized as follows:

- we equip GoI with the ability to capture *choice effects* using a parametric notion of memory structure (Sec. 3);
- we show that the notion of memory structure is able to capture classical, probabilistic and quantum effects (Sec. 3.3);
- we introduce a construction which is *parametric* on the memory, and produces a class of multi-token machines (Sec. 5), proof net systems (Sec. 4) and PCF-style languages (Sec. 6). We prove that (regardless of the specific memory) the multi-token machine is an adequate model of net reduction (Th. 27), and that the nets are an adequate model of PCF term rewriting (Th. 28);
- we develop a general notion of parallel abstract reduction system, which allows us to deal with the combination of parallelism and probabilistic choice in an infinitary setting (Sec. 2).

Being based on a multi-token machine associated to a memory, our model has a concrete nature which makes it well suited to build low-level operational descriptions of higher-order programming languages. In the remainder of this section, we give an informal overview of various aspects of our framework, and motivate with some examples the significance of our contribution.

An extended version of this paper, with proofs, is available [29].

1.1 Geometry of Interaction and Quantum Computation

Geometry of Interaction is interesting as semantics for programming languages [27, 30, 31] because it is a high-level semantics which at the same time is close to low-level implementation and has a clear operational flavor. Computation is interpreted as a flow of information circulating around a network, which essentially is a representation of the underlying program. Computational steps are broken into low-level actions of one or more tokens which are the agents carrying the information around. A long standing open question is whether fully fledged higher-order quantum computation can be modeled operationally via the Geometry of Interaction.

1.1.1 Quantum Computation

As comprehensive references can be found in the literature [2], we only cover the very few concepts that will be needed in this paper. Quantum computation deals with *quantum bits* rather than bits. The state of a quantum system can be represented with a density matrix to account for its probabilistic nature. However for our purpose we shall use in this paper the usual, more operational, non-probabilistic representation. Single quantum bits (or *qubits*) will thus be represented by a ray in a two-dimensional complex Hilbert space, that is, an equivalence class of non-zero vectors up to (complex) scalar multiplication. Information is attached to a qubit by choosing an orthonormal basis $(|0\rangle, |1\rangle)$: a qubit is a superposition of two classical bits (modulo scalar multiplication). If the state of several bits is represented with the product of the states of single bits, the state of a multi-qubit system is represented with the *tensor product* of single-qubit states. In particular, the state of an n-qubit system is a superposition of the state of an n-bit system. We consider superpositions to be normalized.

Two kinds of operations can be performed on qubits. First, one can perform reversible, *unitary gates*: they are unitary maps in the corresponding Hilbert space. A more exotic operation is the *measurement*, which is the only way to retrieve a classical bit out of a quantum bit. This operation is probabilistic: the probabilities depend on the state of the system. Moreover, it modifies the state of the memory. Concretely, if the original memory state is $\alpha_0|0\rangle\otimes\phi_0+\alpha_1|1\rangle\otimes\phi_1$ (with ϕ_0 and ϕ_1 normalized), measuring the first qubit would answer x with probability $|\alpha_x|^2$, and the memory is turned into $|x\rangle \otimes \phi_x$. Note how the measurement not only modifies the measured qubit, but also collapses the global state of the memory.

The effects of measurements are counterintuitive especially in *entangled* system: consider the 2-qubit system $\frac{\sqrt{2}}{2}(|00\rangle + |11\rangle)$. This system is entangled, meaning that it cannot be written as $\phi\otimes\psi$ with 1-qubit states ϕ and ψ. One can get such a system from the state $|00\rangle$ by applying first an *Hadamard gate* H on the second qubit, sending $|0\rangle$ to $\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $\frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)$, therefore getting the state $\frac{\sqrt{2}}{2}(|00\rangle+|01\rangle)$, and then a CNOT (*controlled-not*) gate, sending $|xy\rangle$ to $|x \oplus y\rangle \otimes |y\rangle$. Measuring the first qubit will collapse the entire system to $|00\rangle$ or $|11\rangle$, with equal probability $\frac{1}{2}$.

Remark 1. Notwithstanding the global collapse induced by the measurement, the operations on physically disjoint quantum states are commutative. Let A and B be two quantum states. Let U act on A and V act on B (whether they are unitaries, measurements, or combinations thereof). Consider now $A\otimes B$: applying U on A then V on B is equivalent to first applying V on B and then A on U. In other words, the order of actions on physically separated quantum systems is irrelevant. We use this property in Sec. 3.3.3.

1.1.2 Previous Attempts and Missing Features

A first proposal of Geometry of Interaction for quantum computation is [21]. Based on a purely categorical construction [32], it features duplication but not general entanglement: entangled qubits cannot be separately acted upon. As the authors recognize, a limit of their approach is that their GoI is single-token, and they already suggest that using several tokens could be the solution.

Example 2. As an example, if $S = \frac{\sqrt{2}}{2}(|00\rangle + |11\rangle)$, the term

$$\texttt{let}\, x\otimes y = S\, \texttt{in}\, (Ux)\otimes(Vy) \tag{1}$$

cannot be represented in [21], because it is not possible to send entangled qubits to separate parts of the program.

A more recent proposal [25], which introduces an operational semantics based on *multi-tokens*, can handle general entanglement. However, it does neither handle duplication nor recursion. More than that, the approach relies on termination to establish its results, which therefore do not extend to an infinitary setting: it is not enough to "simply add" duplication and fix points.

Example 3. In [25] it is not possible to simulate the program that tosses a coin (by performing a measurement), returns a fresh qubit on head and repeats on tail. In mock-up ML, this program becomes

$$\texttt{letrec}\, f\, x = (\texttt{if}\, x\, \texttt{then}\, \texttt{new}\, \texttt{else}\, f\, (\mathsf{H}\, \texttt{new}))\, \texttt{in}\, (f\, (\mathsf{H}\, \texttt{new}))$$

where `new` creates a fresh qubit in state $|0\rangle$ and where the `if` test performs a measurement on the qubit x. Note how the measure of

$\mathsf{H}\,\mathtt{new}$ amounts to tossing a fair coin: $\mathsf{H}\,\mathtt{new}$ produces $\frac{\sqrt{2}}{2}(|0\rangle+|1\rangle)$. Measuring gives $|0\rangle$ and $|1\rangle$ with probability $\frac{1}{2}$.

Example 3 will be our leading example all along the paper. Furthermore, we shall come back to both examples in Sec. 7.1.1.

1.2 Parallel Choices: Confluence and Termination

When dealing with both probabilistic choice and infinitary reduction, parallelism makes the study of confluence and convergence highly non-trivial. The issue of confluence arises as soon as choices and duplication are both available, and non-termination adds to the challenges. Indeed, it is easy to see how tossing a coin and duplicating the result does not yield the same probabilistic result as tossing twice the coin. To play with this, let us take for example the following term of the probabilistic λ-calculus [33]: $M = (\lambda x.x \;\mathtt{xor}\; x)((\mathtt{tt} \oplus \mathtt{ff}) \oplus \Omega)$ where $\mathtt{tt}$ and $\mathtt{ff}$ are boolean constants, Ω is a divergent term, $\oplus$ is the choice operator (here, tossing a fair coin), and $\mathtt{xor}$ is the boolean operator computing the exclusive or. Depending on which of the two redexes we fire first, M will evaluate to either the distribution $\{\mathtt{ff}^{\frac{1}{2}}\}$ or to the distribution $\{\mathtt{tt}^{\frac{1}{8}}, \mathtt{ff}^{\frac{1}{8}}\}$. In ordinary, deterministic PCF, any program of boolean type may or may not terminate, depending on the reduction strategy, but its normal form, if it exists, is unique. This is not the case for our probabilistic term M: depending on the choice of the redex, it evaluates to two distributions which are simply not comparable.

In the case of probabilistic λ-calculi, the way-out to this is to fix a reduction strategy; the issue however is not only in the syntax, it appears—at a more fundamental level—also in the model. This is the case for [33], where the model itself does not support parallel probabilistic choice. Similarly, in the development of a Game Semantics or Geometry of Interaction model for probabilistic λ-calculi, the standard approach has been to use a polarized setting, so to impose a strict form of sequentiality [34, 35]. If instead we choose to have parallelism in *the model*, confluence is not granted and even the *definition* of convergence is non-trivial.

In this paper we propose a probabilistic model that is *infinitary and parallel but confluent*; to achieve this, in Sec. 2 we develop some results which are general to any probabilistic abstract reduction system, and which to our knowledge are novel. More specifically, we provide sufficient conditions for an infinitary probabilistic system to be confluent and to satisfy a property which is the probabilistic analogue of the familiar "weak normalization implies strong normalization". We then show that the parametric models which we introduce (both the proof nets and the multi-token machine) satisfy this property; this is indeed what ultimately grants the adequacy results.

1.3 Overview of the Framework, and Its Advantages

A quantum program has on the one hand features which are specific to quantum computing, and on the other hand standard constructs. This is indeed the case for many paradigmatic languages; analyzing the features separately is often very useful. Our framework clearly separates (both in the language and in its operational model) the constructs which are common to all programming languages (e.g. recursion) and the features which are specific to some of them (e.g. measurement or probabilistic choice). The former is captured by a *fixed operational core*, the latter is encapsulated within a *memory structure*. This approach has two distinctive advantages:

- *Facilitate Comparison between Different Languages*: clearly separating in the semantics the standard features from the "notions of computation" which is the specificity of the language, allows for an easier comparison between different languages.
- *Simplify the Design of a Language with Its Operational Model*: it is enough to focus on the memory structure which encapsulates the desired effects. Once such a memory structure is given, the construction provides an adequate Geometry of Interaction model for a PCF-like language equipped with that memory.

Memory structures are introduced in Sec. 3, while the operational core is based on Linear Logic: we have a *linearly-typed* PCF-like language, *Geometry of Interaction*, and its syntactical counterpart, *proof nets*. Proof nets are a graph-based formal system that provides a powerful tool to analyze the execution of terms as a rewriting process which is mostly parallel, local, and asynchronous.

More in detail, our framework consists of:

1. a notion of *memory structure*, whose operations are suitable to capture a range of choice effects;
2. an operational core, which is articulated in the *three rewrite systems* (a proof net system, a GoI multi-token machine, and a PCF-style language);
3. a construction which is *parametric on the memory*, and lifts each base rewrite system into a more expressive operational system. We respectively call these systems: program nets, MSIAM machines and $\mathsf{PCF}_{\mathsf{AM}}$ abstract machines.

Finally, the three forms of systems are *all related by adequacy results*. As long as the memory operations satisfy commutativity, the construction produces an adequate GoI model for the corresponding PCF language. More precisely, we prove— again *parametrically on the memory*—that the MSIAM is an adequate model of program net reduction (Th. 27), and program nets are expressive enough to adequately represent the behavior of the PCF language (Th. 28).

1.4 Related Work

The low-level layer of our framework can be seen as a generalization and a variation of systems which are in the literature. The nets and multi-token machine we use are a variation of those from [28], the linearly typed PCF language is the one in [13] (minus lists and coproducts). What we add in this paper are the right tools to deal with challenges like probabilistic parallel reduction and entanglement. Neither quantum nor probabilistic choice can be treated in [28], because of the issues we clarified in Sec. 1.2. The specificity of our proposal is really its ability to deal with *choice* together with *parallelism*.

We already discussed previous attempts to give a GoI model of quantum computation, and their limits, in Sec. 1.1.2 above. Let us quickly go through other models of quantum computation. Our parametric memory is presented equationally: equational presentations of quantum memory are common in the literature [36, 37]. Other models of quantum memories are instead based on Hilbert spaces and completely positive maps, as in [13, 20]. In both of these approaches, the model captures with precision the structure and behavior of the memory. Instead, in our setting, we only consider the interaction between the memory and the underlying computation by a set of equations on the state of the memory at a given address, the allocation of fresh addresses, and the low-level actions.

Finally, taking a more general perspective, our proposal is by no means the first one to study effects in an interactive setting. Dynamic semantics such as GoI and Game Semantics are gaining interest and attention as semantics for programming languages because of their operational flavor. [35, 38, 39] all deal with effects in GoI. A common point to all these works is to be single-token. While our approach at the moment only deals with choice effects, we indeed deal with *parallelism*, a challenging feature which was still missing.

2. PARS: Probabilistic Abstract Reduction Systems

Parallelism allows critical pairs; as we observed in Sec. 1.2, firing different redexes will produce different distributions and can lead to possibly very different results. Our parallel model however enjoys a property similar to the diamond property of abstract reduction systems. Such a property entails a number of important consequences for confluence and normalization, and these results in fact are general to any probabilistic abstract reduction system. In particular, we define what we mean by strong and weak normalization in a probabilistic setting, and we prove that *a suitable adaptation* of the diamond property guarantees confluence and a form of *uniqueness of normal forms*, not unlike what happens in the deterministic case. Th. 10 is the main result of the section.

In a probabilistic context, spelling out the diamond property requires some care. We will introduce a strongly controlled notion of reduction on distributions ($\rightrightarrows$). The need for this control has the same roots as in the deterministic case: please recall that strong normalization follows from weak normalization by the diamond property ($b \leftarrow a \rightarrow c \Rightarrow b = c \vee \exists d(b \rightarrow d \leftarrow c)$) but *not* from subcommutativity ($b \leftarrow a \rightarrow c \Rightarrow \exists d(b \rightarrow^{=} d \leftarrow^{=} c)$) which appears very similar, but "leaves space" for an infinite branch.

2.1 Distributions and PARS

We start by setting the basic definitions. Given a set A, we note $DST(A)$ for the set of *probability distributions* on A: any $\mu \in DST(A)$ is a function from A to $[0,1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. A distribution μ is *proper* if $\sum_{a \in A} \mu(a) = 1$. We indicate with $SUPP(\mu)$ the support of a distribution μ, *i.e.* the subset of A whose image under μ is not 0. On $DST(A)$, we define the relation $\subseteq$ point-wise: $\mu \subseteq \rho$ if $\mu(a) \leq \rho(a)$ for each $a \in A$.

A *probabilistic abstract reduction system (PARS)* is a pair $\mathcal{A} = (A, \rightarrow)$ consisting of a set A and a relation $\rightarrow \subseteq A \times DST(A)$ (rewrite relation, or reduction relation) such that for each $(a, \mu) \in \rightarrow$, $SUPP(\mu)$ is finite. We write $a \rightarrow \mu$ for $(a, \mu) \in \rightarrow$. An element $a \in A$ is *terminal* or in *normal form* (w.r.t. $\rightarrow$) if there is no μ with $a \rightarrow \mu$, which we write $a \not\rightarrow$.

We can partition any distribution μ into a distribution μ° on terminal elements, and a distribution $\bar{\mu}$ on elements for which there exists a reduction, as follows:

$$\mu^{\circ}(a) = \begin{cases} \mu(a) & \text{if } a \not\rightarrow, \\ 0 & \text{otherwise;} \end{cases} \qquad \bar{\mu}(a) = \mu(a) - \mu^{\circ}(a).$$

The *degree of termination* of μ, written $\mathcal{T}(\mu)$, is $\sum_{a \in A} \mu^{\circ}(a)$.

We write $\looparrowright$ for the *reflexive and transitive closure* of $\rightarrow$, namely the smallest subset of $A \times DST(A)$ closed under the following rules: if $a \rightarrow \mu$ then $a \looparrowright \mu$; we always have $a \looparrowright \{a^1\}$; whenever $a \looparrowright \mu + \{b^p\}$, $b \looparrowright \rho$ and $b \notin SUPP(\mu)$ we have $a \looparrowright \mu + p \cdot \rho$.

The Relation $\rightrightarrows$. In order to extend to PARS classical results on termination for rewriting systems, we define the binary relation $\rightrightarrows$, which lifts the notion of one step reduction to distributions: we require that *all* non-terminal elements are indeed reduced. The relation $\rightrightarrows \subseteq DST(A) \times DST(A)$ is defined as

$$\frac{\mu = \mu^{\circ} + \bar{\mu} \quad \{a \rightarrow \rho_a\}_{a \in SUPP(\bar{\mu})}}{\mu \rightrightarrows \mu^{\circ} + \sum_{a \in SUPP(\bar{\mu})} \mu(a) \cdot \rho_a} \ .$$

Please note that in the derivation above, we require $a \rightarrow \rho_a$ *for each* $a \in SUPP(\bar{\mu})$. Observe also that $\mu^{\circ} \rightrightarrows \mu^{\circ}$ since $SUPP(\bar{\mu^{\circ}}) = \emptyset$. We write $\mu \rightrightarrows^{n} \rho$ if μ reduces to ρ in $n \geq 0$ steps; we write $\mu \rightrightarrows^{*} \rho$ if there is any *finite* sequence of reductions from μ to ρ.

With a slight abuse of notation, in the rest of the paper we sometime write $\{a\}$ for $\{a^1\}$, or simply a when clear from the context. As an example, we write $a \rightrightarrows \mu$ for $\{a^1\} \rightrightarrows \mu$. Moreover, the distribution $\{a_1^{p_1}, \ldots, a_n^{p_n}\}$ will be often indicated as $\sum p_i \cdot \{a_i\}$ thus facilitating algebraic manipulations.

2.2 Normalization and Confluence

In this subsection, we look at normalization and confluence in the probabilistic setting, which we introduced in Sec. 2.1. We need to distinguish between *weak* and *strong* normalization. The former refers to the *possibility* to reach normal forms following any reduction order, while the latter (also known as *termination*, see [40]) refers to the *necessity* of reaching normal forms. In both cases, the concept is inherently quantitative.

Definition 4 (Weak and Strong Normalization)**.** Let $p \in [0,1]$ and let $\mu \in DST(A)$ Then:

- μ *weakly p-normalizes* (or weakly normalizes with probability at least p) if there exists ρ such that $\mu \rightrightarrows^{*} \rho$ and $\mathcal{T}(\rho) \geq p$.
- μ *strongly p-normalizes* (or strongly normalizes with probability at least p) if there exists n such that $\mu \rightrightarrows^{n} \rho$ implies $\mathcal{T}(\rho) \geq p$, for all ρ.

The relation $\rightarrow$ is said *uniform* if for each p, and each $\mu \in DST(A)$, weak p-normalization implies strong p-normalization.

Even the mere notion of convergent computation must be made quantitative here:

Definition 5 (Convergence)**.** The distribution $\mu \in DST(A)$ *converges with probability p*, written $\mu \Downarrow_p$, if $p = \sup_{\mu \rightrightarrows^{*} \rho} \mathcal{T}(\rho)$.

Observe that for every μ there is a unique probability p such that $\mu \Downarrow_p$. Please also observe how Definition 5 is taken over all ρ such that $\mu \rightrightarrows^{*} \rho$, thus being forced to take into account all possible reduction orders. If $\rightarrow$ is uniform, however, we can reach the limit along *any* reduction order:

Proposition 6. *Assume $\rightarrow$ is uniform. Then for every μ such that $\mu \Downarrow_p$ and for every sequence of distributions $(\rho_n)_n$ such that $\mu = \rho_0$ and $\rho_n \rightrightarrows \rho_{n+1}$ for every n, it holds that $p = \lim_{n \to \infty} \mathcal{T}(\rho_n)$.* □

A PARS is said to be confluent iff $\rightrightarrows$ is a confluent relation in the usual sense:

Definition 7 (Confluence)**.** The PARS $(A, \rightarrow)$ is said to be *confluent* if whenever $\tau \rightrightarrows^{*} \mu$ and $\tau \rightrightarrows^{*} \xi$, there exists ρ such that $\mu \rightrightarrows^{*} \rho$ and $\xi \rightrightarrows^{*} \rho$.

2.3 The Diamond Property in a Probabilistic Scenario

We now study a property which insures confluence and uniformity.

Definition 8 (Diamond Property for PARS)**.** A PARS $(A, \rightarrow)$ *satisfies the diamond property* if the following holds. Assume $\mu \rightrightarrows \nu$ and $\mu \rightrightarrows \xi$. Then (1) $\nu^{\circ} = \xi^{\circ}$ and (2) $\exists \rho$ with $\nu \rightrightarrows \rho$ and $\xi \rightrightarrows \rho$.

As an immediate consequence:

Corollary 9 (Confluence)*.* *If $(A, \rightarrow)$ satisfies the diamond property, then $(A, \rightarrow)$ is confluent.* □

Finally, then, the diamond property ensures that weak p-normalization implies strong p-normalization, precisely like for usual abstract reduction systems:

Theorem 10 (Normalization and Uniqueness of Normal Forms)**.** *Assume $(A, \rightarrow)$ satisfies the diamond property. Then:*

1. ***Uniqueness of normal forms.*** *$\mu \rightrightarrows^{k} \rho$ and $\mu \rightrightarrows^{k} \tau$ for some $k \in \mathbb{N}$ implies $\rho^{\circ} = \tau^{\circ}$.*
2. ***Uniformity**. If μ is weakly p-normalizing (for some $p \in [0,1]$), then μ strongly p-normalizes,* i.e.*, $\rightarrow$ is uniform.*

Proof. First note that (2) follows from (1). In order to prove (1), we use an adaptation of the familiar "tiling" argument. It is not exactly the standard proof because reaching some normal forms in a distribution is not the end of a sequence of reductions. Assume $\mu = \rho_0 \rightrightarrows \rho_1 \rightrightarrows \ldots \rightrightarrows \rho_k$, and $\mu \rightrightarrows \tau_1 \rightrightarrows \ldots \rightrightarrows \tau_k$. We prove $\rho_k^{\circ} = \tau_k^{\circ}$ by induction on k. If $k = 1$, the claim is true by Definition 8 (1). If $k > 1$ we tile (w.r.t. $\rightrightarrows$), as depicted below:

$$\begin{tikzcd}
 & \tau_1 \arrow[r] \arrow[dr] & \tau_2 \arrow[r, dash, dotted] & \cdots \arrow[r] & \tau_k \\
\mu \arrow[ur] \arrow[dr] & & \sigma_1 \arrow[dr] & & \\
 & \rho_1 \arrow[ur] \arrow[dr] & & \sigma_2 \arrow[dr, dash, dotted] & \\
 & & \rho_2 \arrow[ur] \arrow[dr, dash, dotted] & & \sigma_{k-1} \\
 & & & \rho_{k-1} \arrow[ur] \arrow[r] & \rho_k
\end{tikzcd}$$

we build the sequence $\sigma_0 = \tau_1 \rightrightarrows \sigma_1 ... \rightrightarrows \sigma_{k-1}$ (see the Figure on the side) where each σ_{i+1} ($i \geq 0$) is obtained via Definition 8 (2), from $\rho_i \rightrightarrows \rho_{i+1}$ and $\rho_i \rightrightarrows \sigma_i$, by closing the diamond. By Definition 8 (1) $\rho_k^\circ = \sigma_{k-1}^\circ$. Now we observe that $\tau_1 \rightrightarrows^{k-1} \tau_k$ and $\tau_1 \rightrightarrows^{k-1} \sigma_{k-1}$. Therefore we have (by induction) $\sigma_{k-1}^\circ = \tau_k^\circ$, from which we conclude $\rho_k^\circ = \tau_k^\circ$. $\square$

3. Memory Structures

In this section we introduce the notion of a memory structure. Commutativity of the memory operations is ensured by a set of equations. To deal with the notion of fresh addresses, and avoid unnecessary bureaucracy, it is convenient to rely on nominal sets. The basic definitions are recalled below (for details, see, *e.g.*, [41]).

3.1 Nominal Sets

If G is a group, then a *G-set* $(M, \cdot)$ is a set M equipped with an action of G on M, *i.e.* a binary operation $(\cdot) : G \times M \longrightarrow M$ which respects the group operation. Let I be a countably infinite set; let M be a set equipped with an action of the group $\mathrm{Perm(I)}$ of *finitary permutations* of I. A *support* for $m \in M$ is a subset $A \subseteq I$ such that for all $\sigma \in \mathrm{Perm(I)}, \forall i \in A, \sigma i = i$ implies $\sigma \cdot m = m$. A *nominal set* is a $\mathrm{Perm(I)}$-set all of whose elements have finite support. In this case, if $m \in M$, we write $\mathrm{supp}(m)$ for the smallest support of m. The complementary notion of support is freshness: $i \in I$ is *fresh* for $m \in M$ if $i \notin \mathrm{supp}(m)$. We write $(i\ j)$ for the transposition which swaps i and j.

We will make use of the following characterization of support in terms of transpositions: $A \subseteq I$ supports $m \in M$ if and only if for every $i, j \in I - A$ it holds that $(i\ j) \cdot m = m$. As a consequence, for all $i, j \in I$, if they are fresh for $m \in M$ then $(i\ j) \cdot m = m$.

3.2 Memory Structures

A *memory structure* $\mathrm{Mem} = (\mathrm{Mem}, I, \cdot, \mathcal{L})$ consists of an infinite, countable set I whose elements $i, j, k, \ldots$ we call *addresses*, a nominal set $(\mathrm{Mem}, \cdot)$ each of whose elements we call *memory states*, or more shortly, *memories*, and a finite set $\mathcal{L}$ of *operations*.

We write I^* for the set of all tuples made from elements of I. A tuple is denoted with $(i_1, \ldots, i_n)$, or with $\vec{i}$. To a memory structure are associated the following maps.

- $\mathrm{test} : I \times \mathrm{Mem} \to DST(Bool \times \mathrm{Mem})$ (Observe that the set Mem might be updated by the operation test: for this reason, it also appears in the codomain – See Remark 14),
- $\mathrm{update} : I^* \times \mathcal{L} \times \mathrm{Mem} \rightharpoonup \mathrm{Mem}$ (partial map),
- $\mathrm{arity} : \mathcal{L} \to \mathbb{N}$,

and the following three properties.

(1) The maps test and update respect the group action:

- $\sigma \cdot (\mathrm{test}(i, m)) = \mathrm{test}(\sigma(i), \sigma \cdot m)$,
- $\sigma \cdot (\mathrm{update}(\vec{i}, x, m)) = \mathrm{update}(\sigma(\vec{i}), x, \sigma \cdot m)$,

where the action of $\mathrm{Perm(I)}$ is extended in the natural way to distributions and pairing with booleans.

(2) The action of a given operation on the memory is only defined for the correct arity. More precisely, $\mathrm{update}((i_1 \ldots i_n), x, m)$ is defined if and only if the i_k's are pairwise disjoint and $\mathrm{arity}(x) = n$.

(3) Disjoint tests and updates commute: assume that $i \neq j$, that j does not meet $\vec{k}$, and that $\vec{k}$ and $\vec{k'}$ are disjoint. First, updates on $\vec{k}$ and $\vec{k'}$ commute: $\mathrm{update}(\vec{k}, x, \mathrm{update}(\vec{k'}, x', m)) = \mathrm{update}(\vec{k'}, x', \mathrm{update}(\vec{k}, x, m))$. Then, tests on i commute with tests on j and tests of j commute with updates on $\vec{k}$. We pictorially represent these equations in Fig. 1. The drawings are meant to be read from top to bottom and represent the successive memories along action. Probabilistic behavior is represented with two exiting arrows, annotated with their respective probability of occurrence, and the boolean resulting from the test operation. Intermediate memories are unnamed and represented with "$\cdot$".

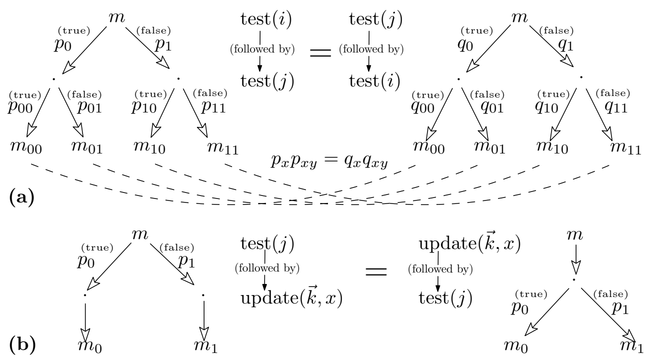

Figure 1. Commutation of Tests and Updates.

3.3 Instances of Memory Structures

The structure of memory is flexible and can accommodate several choice effects. Let us give some relevant examples. Typically, here I is $\mathbb{N}$, but any countable set would do the job.

3.3.1 Deterministic, Integer Registers

The simplest instance of memory structure is the deterministic one, corresponding to the case of classical PCF (this subsumes, in particular, the case studied in [28]). Memories are simply functions $\mathbf{m}$ from I to $\mathbb{N}$, of value 0 apart for a finite subset of I. The test on address i is deterministic, and tests whether $\mathbf{m}(i)$ is zero or not. Operations in $\mathcal{L}$ may include the unary predecessor and successor, and for example the binary max operator.

Example 11. A typical representation of this deterministic memory is a sequence of integers: indexes correspond to addresses and coefficients to values. A completely free memory is for example the sequence $\mathbf{m}_0 = (0, 0, 0, \ldots)$. If S corresponds to the successor and P to the predecessor, here is what happens to the memory for some operations. The memory $\mathbf{m}_1 := \mathrm{update}(0, S, \mathbf{m}_0)$ is $(1, 0, 0, 0, \ldots)$, the memory $\mathbf{m}_2 := \mathrm{update}(1, S, \mathbf{m}_1)$ is $(1, 1, 0, 0, \ldots)$, and the memory $\mathbf{m}_3 := \mathrm{update}(0, P, \mathbf{m}_2)$ is $(0, 1, 0, 0, \ldots)$. Finally, $\mathrm{test}(1, \mathbf{m}_3) = (\mathrm{false}, (0, 1, 0, 0, \ldots))$. Note that we do not need to keep track of an infinite sequence: a dynamic, finite list of values would be enough. We'll come back to this in Sec. 3.3.3.

Remark 12. The equations on memory structures enforce the fact that all fresh addresses (*i.e.*, not on the support of the nominal set) have equal values. Note that however the conditions do not impose any particular "default" value. These equations also state that, in the deterministic case, a test action on i can only modify the memory at address i. Otherwise, it could for example break the commutativity of update and test (unless $\mathcal{L}$ contains trivial operations, only).

3.3.2 Probabilistic, Boolean Registers

When the test operator is allowed to have a genuinely probabilistic behavior, the memory model supports the representation of probabilistic boolean registers. In this case, a memory $\mathbf{m}$ is a function from I to the real interval $[0, 1]$, whose values represent probabilities of observing "true". The test on address i could return

$$\mathbf{m}(i)\{(\mathrm{true}, \mathbf{m}\{i \mapsto 1\})\} + (1 - \mathbf{m}(i))\{(\mathrm{false}, \mathbf{m}\{i \mapsto 0\})\}$$

Operations in $\mathcal{L}$ may for example include a unary "coin flipping" operation setting the value associated to i to some fixed probability.

Example 13. If as in Example 11 we represent the memory as a sequence, a memory filled with the value "false" would be $\mathbf{m}_0 = (0, 0, 0, \ldots)$. Assume c is the unary operation placing a fair coin at the corresponding address; if $\mathbf{m}_1$ is $\mathrm{update}(0, c, \mathbf{m}_0)$, we have $\mathbf{m}_1 = (\frac{1}{2}, 0, 0, 0, \ldots)$. Then $\mathrm{test}(0, \mathbf{m}_1)$ is the distribution $\frac{1}{2}(\mathrm{false}, (0, 0, 0, 0, \ldots)) + \frac{1}{2}(\mathrm{true}, (1, 0, 0, 0, \ldots))$.

3.3.3 Quantum Registers

A standard model for quantum computation is the QRAM model: quantum data is stored in a memory seen as a list of (quantum) registers, each one holding a qubit which can be acted upon. The model supports three main operations: creation of a new register, measurement of a register, and application of unitary gates on one or more registers, depending on the arity of the gate under scrutiny. This model has been used extensively in the context of quantum lambda-calculi [11, 13, 24], with minor variations. The main choice to be made is whether measurement is destructive (*i.e.*, if one uses garbage collection) or not (*i.e.*, the register is not reclaimed).

A Canonical Presentation of Quantum Memory. To fix things, we shall concentrate on the presentation given in [13]. We briefly recall it. Given n qubits, a memory is a normalized vector in $(\mathbb{C}^2)^{\otimes n}$ (equivalent to a ray). A linking function maps the position of each qubit in the list to some pointer name. The creation of a new qubit turns the memory $\phi \in (\mathbb{C}^2)^{\otimes n}$ into $\phi \otimes |0\rangle \in (\mathbb{C}^2)^{\otimes(n+1)}$. The measurement is destructive: if $\phi = \alpha_0 q_0 + \alpha_1 q_1$, where each q_b (with $b = 0, 1$) is normalized of the form $\sum_i \phi_{b,i} \otimes |b\rangle \otimes \psi_{b,i}$, then measuring ϕ returns $\sum_i \phi_{b,i} \otimes \psi_{b,i}$ with probability $|\alpha_b|^2$. Finally, the application of a k-ary unitary gate U on $\phi \in (\mathbb{C}^2)^{\otimes n}$ simply applies the unitary matrix corresponding to U on the vector ϕ. The language comes with a chosen set $\mathcal{U}$ of such gates.

Quantum Memory as a Nominal Set. The quantum memory can be equivalently presented using a memory structure: in the following we shall refer to it as $\mathcal{Q}$. The idea is to use nominal set to make precise the hand-waved "pointer name", and formalize the idea of having a finite core of "in use" qubits, together with an infinite pool of fresh qubits. Let $\mathcal{F}_0$ be the set of (set-)maps from I (the infinite, countable set of Sec. 3.2) to $\{0, 1\}$ that have value 0 everywhere except for a finite subset of I. We have a *memory structure* as follows.

I is the domain of the set-maps in $\mathcal{F}_0$. The *nominal set* $(\mathrm{Mem}, \cdot)$ is defined with $\mathrm{Mem} = \mathcal{H}_0$, *i.e.* the Hilbert space built from finite (complex) linear combinations over $\mathcal{F}_0$, while the *group action* $(\cdot)$ corresponds to permutation of addresses: $\sigma \cdot \mathbf{m}$ is simply the function-composition of σ with the elements of $\mathcal{F}_0$ in superposition in $\mathbf{m}$. The support of a particular memory $\mathbf{m}$ is finite: it consists of the set of addresses that are not mapped to 0 by some (set)-function in the superposition. The *set of operations* $\mathcal{L}$ is the chosen set $\mathcal{U}$ of unitary gates. The *arity* is the arity of the corresponding gate.

Finally, the *update* and *test* operations correspond respectively to the application of an unitary gate, and to a measurement followed by a (classical) boolean test on the result. We omit the formalization of these operations in the nominal set setting; instead we show how this presentation in terms of nominal sets is equivalent to the previous more canonical one.

Equivalence of the Two Presentations. Let $\mathbf{m} \in \mathcal{Q}$. We can always consider a finite subset of I, say $I_0 = \{i_0 \ldots i_n\}$ for some integer n such that all other addresses are fresh. As fresh values are 0 in $\mathbf{m}$, then $\mathbf{m}$ is a superposition of sequences that are equal to 0 on $I \setminus I_0$. Then $\mathbf{m}$ can be represented as "$\phi \otimes |000\ldots\rangle$" for some (finite) vector ϕ. We can omit the last $|0000\ldots\rangle$ and only work with the vector ϕ: we are back to the canonical presentation of quantum memory. Update and test can then be defined on the nominal set presentation through this equivalence.

Equations. Memory structures come also with equations, which are indeed satisfied. Referring to Sec. 3.2 : (1) is simply renaming of qubits, (2) is a property of applying a unitary, and (3) holds because of the equations corresponding to the tensor of two unitaries or the tensor of a unitary and a measurement (see Remark 1).

Remark 14. The quantum case makes clear why Mem appears in the codomain of test: since in general the measurement of a register collapses the global state of the memory (see Sec. 1.1.1), the modified memory has to be returned together with the result.

3.4 Overview of the Forthcoming Sections

We use memory structures to encapsulate effects in three different settings. In Sec. 4, we enrich proof nets with a memory, in Sec. 5, we enrich token machines with a memory, while in Sec. 6, we equip PCF terms with a memory. The construction is uniform for all the three systems, to which we refer as *operational systems*, as opposite to the *base rewrite systems* on top of which we build (see Sec. 1.3).

4. Program Nets and Their Dynamics

In this section, we introduce *program nets*. The base rewrite system on which they are built is a variation[1] of SMEYLL nets, as introduced in [28]. SMEYLL nets are MELL (Multiplicative Exponential Linear Logic) proof nets extended with *fixpoints* (Y-boxes) which model recursion, *additive boxes* ($\perp$-boxes) which capture the if-then-else construct, and a family of *sync nodes*, introducing explicit synchronization points in the net.

The *novelty* of this section is the operational system which we introduce in Sec. 4.3, by means of our parametric construction: given a memory structure and SMEYLL nets, we define program nets and their reduction. We prove that program nets are a PARS which satisfies the diamond property, and therefore confluence and uniqueness of normal forms both hold. Program nets also satisfy cut elimination, *i.e.* deadlock-freeness of nets rewriting.

4.1 Formulas

The language of *formulas* is the same as for MELL. In this paper, we restrict our attention to the constant-only fragment, *i.e.*:

$$A ::= 1 \mid \perp \mid A \otimes A \mid A \parr A \mid !A \mid ?A.$$

The constants $1, \perp$ are the *units*. As usual, linear negation $(\cdot)^\perp$ is extended into an involution on all formulas: $A^{\perp\perp} \equiv A$, $1^\perp \equiv \perp$, $(A \otimes B)^\perp \equiv A^\perp \parr B^\perp$, $(!A)^\perp \equiv ?A^\perp$. Linear implication is a defined connective: $A \multimap B \equiv A^\perp \parr B$. *Positive formulas* P and *negative formulas* N are respectively defined as: $P ::= 1 \mid P \otimes P$, and $N ::= \perp \mid N \parr N$.

4.2 SMEYLL Nets

A SMEYLL *net* is a pre-net (*i.e.* a well-typed graph) which fulfills a *correctness criterion*.

Pre-Nets. A *pre-net* is a labeled *directed* graph R built over the alphabet of nodes represented in Fig. 2.
Edges. Every edge in R is labeled with a formula; the label of an edge is called its *type*. We call those edges represented below (resp. above) a node symbol *conclusions* (resp. *premises*) of the node. We will often say that a node "has a conclusion (premise) A" as shortcut for "has a conclusion (premise) of type A". When we need more precision, we explicitly distinguish an edge and its type and we use variables such as e, f for the edges. Each edge is a conclusion of exactly one node and is a premise of at most one node. Edges which are not premises of any node are called the *conclusions of the net*.

[1] In this paper, reduction of the $\perp$-box is not deterministic; there is otherwise no major difference with [28].

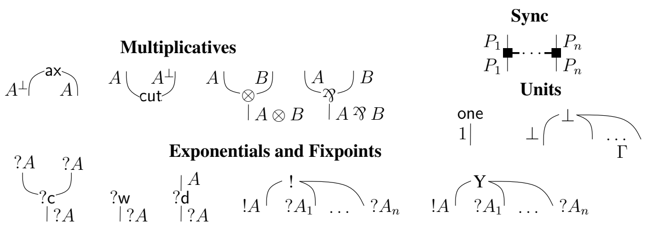

Figure 2. Nodes.

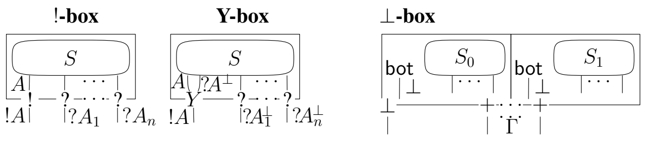

Figure 3. Boxes.

Nodes. The sort of each node induces constraints on the number and the labels of its premises and conclusions. The constraints are graphically shown in Fig. 2. A *sync node* has $n \in \mathbb{N}$ premises of types $P_1, P_2, \cdots, P_n$ respectively and n conclusions of the same types $P_1, P_2, \cdots, P_n$ as the premises, where each P_i is a positive type. A sync node with n premises and conclusions is drawn as n many black squares connected by a line as in the figure. The total number of $\mathbf{1}$'s in the P_i's is called the *arity* of the sync node.

We call *boxes* the nodes $\perp$, !, and Y. The leftmost conclusion of a box is said to be *principal*, while the other ones are *auxiliary*. The node $\perp$ has conclusion $\{\perp, \Gamma\}$ with $\Gamma \neq \emptyset$. The exponential boxes ! and Y have conclusions $\{!A, ?\Gamma\}$ (Γ possibly empty). To each !-box (resp. Y-box) is associated a content, *i.e.* a pre-net of conclusions $\{A, ?\Gamma\}$ (resp. $\{A, ?A^\perp, ?\Gamma\}$). To each $\perp$-box are associated *a left and a right content*: each content is a pair $(\mathtt{bot}, S)$, where bot is a new node that has no premise and one conclusion $\perp$, and S is a pre-net of conclusions Γ. We represent a box b and its content(s) as in Fig. 3. The nodes and edges in the content are said to be *inside* b. As is standard, we often call a crossing of the box border a *door*, which we treat as a node. We then speak of premises and conclusion of the principal (resp. auxiliary) door.

Depth. A node occurs *at depth 0* or *on the surface* in the pre-net R if it is a node of R. It occurs *at depth* $n+1$ in R if it occurs at depth n in a pre-net associated to a box of R.

Nets. A *net* is given by a pre-net R which satisfies the correctness criterion of [28], together with a *total* map $\mathtt{mkname}_R : \mathtt{SyncNode}(R) \to \mathcal{L}$, where $\mathcal{L}$ is a finite set of *names* and $\mathtt{SyncNode}(R)$ is the set of sync nodes appearing in R (including those inside boxes); the map $\mathtt{mkname}_R$ is simply naming the sync nodes. From now on, we write R for the triple $(R, \mathcal{L}, \mathtt{mkname}_R)$.

Reduction Rules. Fig. 4 describes the rewriting rules on nets. Note that the redex in the top row has two possible reduction rules, u_0 and u_1. Note also the y reduction, which captures the recursive behavior of the Y-box as a fixpoint (we illustrate this in the example below.) The metavariables X, X_1, X_2 of Fig. 4 range over $\{!, Y\}$ and are used to uniformly specify reduction rules involving exponential boxes (*i.e.*, X's can be either ! or Y). The reduction of net has two constraints: (1) *surface reduction*, *i.e.* a reduction step applies only when the redex is at depth 0, and (2) *exponential steps are closed*, *i.e.* they only take place when the $!A$ premise of the cut is the principal conclusion of a box with no auxiliary conclusion. We come back on the former in Sec. 7.1.2.

Example. The "skeleton" of the program in Example 3 could be encoded as in the LHS of Fig. 5. The recursive function `f` is

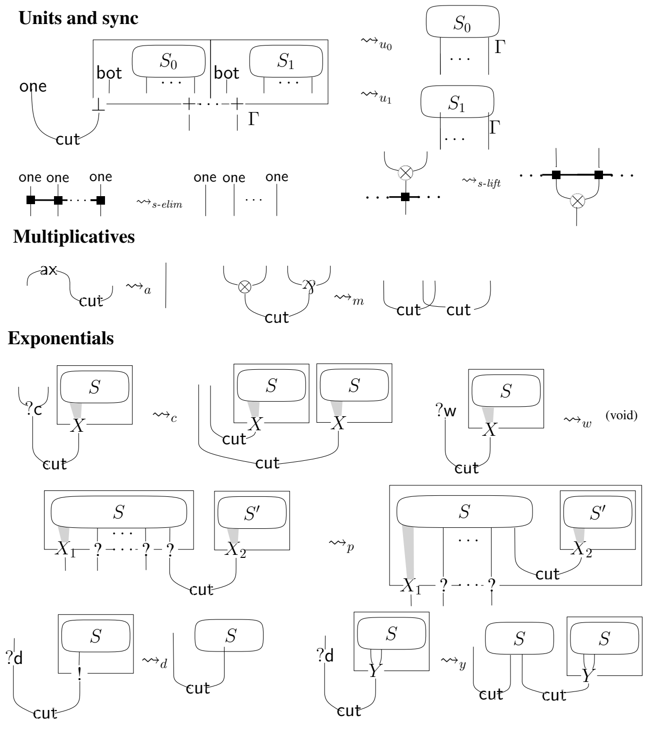

Figure 4. Nets Rewriting Rules.

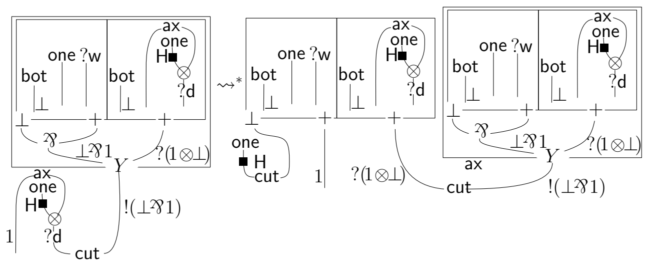

Figure 5. Encoding of Example 3 and Net Rewrite.

represented with a Y-box, and has type $!(1 \multimap 1) = !(\perp ⅋ 1)$. The test is encoded with a $\perp$-box: in one case we forget the function `f` by using ?w and simply return a one node, and in the other case we apply a Hadamard gate, which is represented with a (unary) sync node. To the "`in`" part of the let-rec corresponds the dereliction node ?d, triggering reduction. With the rules presented in the previous paragraph, the net rewrites according to Fig. 5, where the Y-box has been unwound once. From there, we could reduce the one node with the sync node H, but of course doing so we would not handle the quantum memory. In order to associate to reductions an action on the memory, we need a bit more, namely the notion of a program net which is introduced in Sec. 4.3.

4.3 Program Nets

Let $\mathtt{Inputs}(R)$ be the set of all occurrences of $\mathbf{1}$'s which are conclusions of one nodes *at the surface*, and of all the occurrences of $\perp$'s which appear in the conclusions of R.

Definition 15 (Program Nets)**.** Given a memory structure $\mathrm{Mem} = (\mathrm{Mem}, I, \mathcal{L})$, a *raw program net* on Mem is a tuple $(R, \mathtt{ind}_R, \mathbf{m})$ such that

- R is a SMEYLL net (with $\mathtt{mkname}_R : \mathtt{SyncNode}(R) \to \mathcal{L}$),

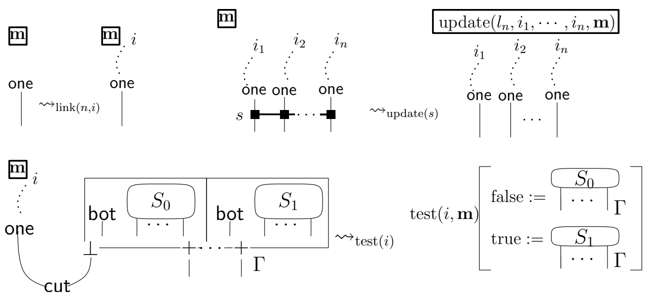

Figure 6. Program Net Reduction Involving Memories.

- $\mathtt{ind}_R : \mathtt{Inputs}(R) \rightharpoonup I$ is an injective *partial* map that is however total on the occurrences of $\perp$,
- $\mathbf{m} \in \mathrm{Mem}$.

We require that the arity of each sync node s matches the arity of $\mathtt{mkname}_R(s)$. Please observe that in the second item in Definition 15, the occurrences of $1's$ belonging to $\mathtt{Inputs}(R)$ are not necessarily in the domain of $\mathtt{ind}_R$; if they are, we say that the corresponding one node is *active*. *Program nets* are the equivalence class of raw program nets over permutation of the indexes. Formally, let $\sigma(R, \mathtt{ind}_R, \mathbf{m}) = (R, \sigma \cdot \mathtt{ind}_R, \sigma \cdot \mathbf{m})$, for $\sigma \in \mathrm{Perm(I)}$. The equivalence class $\mathbf{R} = [(R, \mathtt{ind}_R, \mathbf{m})]$ is $\{\sigma(R, \mathtt{ind}_R, \mathbf{m}) \mid \sigma \in \mathrm{Perm(I)}\}$. We use the symbol $\sim$ for the equivalence relation on raw program nets. $\mathcal{N}$ indicates the set of program nets.

Reduction Rules. We define a relation $\rightsquigarrow \subseteq \mathcal{N} \times DST(\mathcal{N})$, making program nets into a PARS. We first define the relation $\rightsquigarrow$ over raw program nets. Fig. 6 summarizes the reductions in a graphical way; the function $\mathtt{ind}_R$ is represented by the dotted lines.

1. *Link*. If n is a one node of conclusion x, with $\mathtt{ind}_R(x)$ undefined, then $(R, \mathtt{ind}_R, \mathbf{m}) \rightsquigarrow_{\mathrm{link(n,i)}} \{(R, \mathtt{ind}_R \cup \{x \mapsto i\}, \mathbf{m})^1\}$ where $i \in I$ is fresh both in $\mathtt{ind}_R$ and $\mathbf{m}$.
2. *Update*. If $R \rightsquigarrow_s R'$, and s is the sync node in the redex, then
$$(R, \mathtt{ind}_R, \mathbf{m}) \rightsquigarrow_{\mathrm{update(s)}} \{(R', \mathtt{ind}_R, \mathrm{update}(l, \vec{i}, \mathbf{m})^1\}$$
where l is the label of s, and $\vec{i}$ are the addresses of its premises.
3. *Test*. If $R \rightsquigarrow_{u_0} R_0$ and $R \rightsquigarrow_{u_1} R_1$, and i is the address of the premise 1 of the cut, then $(R, \mathtt{ind}_R, \mathbf{m}) \rightsquigarrow_{\mathrm{test(i)}} \mathrm{test}(i, \mathbf{m})[\mathrm{false}{:=}(R_0, \mathtt{ind}_{R_0}, \mathbf{m}),\ \mathrm{true}{:=}(R_1,\ \mathtt{ind}_{R_1}, \mathbf{m})]$, where $\mathtt{ind}_{R_0}$ (resp. $\mathtt{ind}_{R_1}$) is the restriction of $\mathtt{ind}_R$ to $\mathtt{Inputs}(R_0)$ (resp. $\mathtt{Inputs}(R_1)$).
4. Otherwise, if $R \rightsquigarrow_x R'$ with $x \notin \{s, u_0, u_1\}$, then we have
$$(R, \mathtt{ind}_R, \mathbf{m}) \rightsquigarrow_x \{(R', \mathtt{ind}_R, \mathbf{m})^1\}$$

(observe that none of these rules modify the domain of $\mathtt{ind}_R$). The relation $\rightsquigarrow$ extends immediately to program nets (by slight abuse of notation we use the same symbol); Lem. 16 guarantees that the relation is well defined. We write $(R, \mathtt{ind}_R, \mathbf{m}) \overset{r}{\rightsquigarrow} \mu$ for the reduction of the redex r in the raw program net $(R, \mathtt{ind}_R, \mathbf{m})$.

Lemma 16 (Reduction Preserves Equivalence)**.** *Suppose that* $(R, \mathtt{ind}_R, \mathbf{m}) \overset{r}{\rightsquigarrow} \mu$ *and* $(R, \sigma \cdot \mathtt{ind}_R, \sigma \cdot \mathbf{m}) \overset{r}{\rightsquigarrow} \nu$, *then* $\mu \sim \nu$.

Proof. Let us check the rule $\rightsquigarrow_{\mathrm{test}(i)}$. Suppose $(R', \mathtt{ind}_{R'}, \mathbf{m}') = \sigma(R, \mathtt{ind}_R, \mathbf{m})$, $(R, \mathtt{ind}_R, \mathbf{m}) \overset{r}{\rightsquigarrow}_{\mathrm{test}(i)} \mu = \{(R_0, \mathtt{ind}_{R_0}, \mathbf{m}_0)^{p_0}, (R_1, \mathtt{ind}_{R_1}, \mathbf{m}_1)^{p_1}\}$, and $(R', \mathtt{ind}'_R, \mathbf{m}') \overset{r}{\rightsquigarrow}_{\mathrm{test}(i)}$ $\nu = \{(R'_0, \mathtt{ind}'_{R_0}, \mathbf{m}'_0)^{p'_0}, (R'_1, \mathtt{ind}_{R'_1}, \mathbf{m}'_1)^{p'_1}\}$ by reducing the same redex r. It suffices to show that $\nu = \sigma \cdot \mu$. Element-wise, we have to check $R'_i = R_i$, $\mathtt{ind}'_{R_i} = \sigma \circ \mathtt{ind}'_{R_i}$, $\mathbf{m}'_i = \sigma \cdot \mathbf{m}_i$, and $p'_i = p_i$ for $i \in \{0, 1\}$. The first two follow by definition of $\rightsquigarrow_{\mathrm{test}(i)}$ and the last two follow from the equation $\sigma \cdot (\mathrm{test}(i, m)) = \mathrm{test}(\sigma(i), \sigma \cdot m)$. The other rules can be similarly checked. ☐

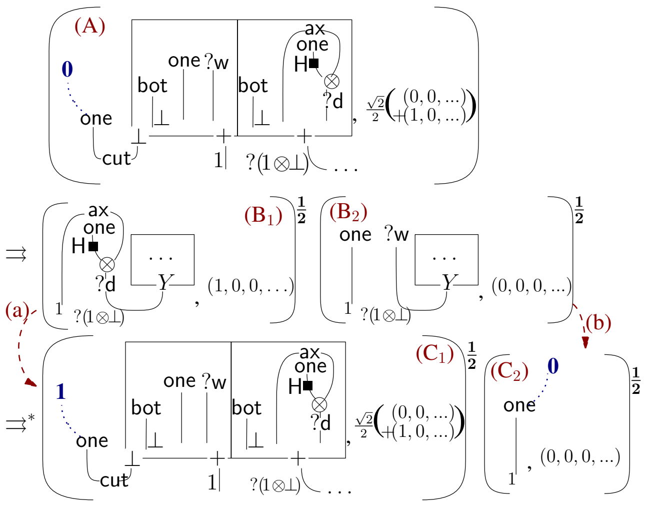

Figure 7. Example of Program Net Rewriting.

Remark 17. In the definition of the reduction rules:
- *Link* is independent from the choice of i. If we chose another address j with the same conditions, then we would have gone to $(R, \mathtt{ind}_R \cup \{x \mapsto j\}, \mathbf{m})$. However this does not cause a problem: by using a permutation $\sigma = (i\,j)$, since $\sigma \cdot \mathbf{m} = \mathbf{m}$ we have $\sigma(R, \mathtt{ind}_R \cup \{x \mapsto i\}, \mathbf{m}) = (R, \mathtt{ind}_R \cup \{x \mapsto j\}, \mathbf{m})$ and therefore $(R, \mathtt{ind}_R \cup \{x \mapsto i\}, \mathbf{m})$ and $(R, \mathtt{ind}_R \cup \{x \mapsto j\}, \mathbf{m})$ are as expected the exact same program net.
- In the rules *Update* and *Test*, the involved one nodes are required to be active.

The pair $(\mathcal{N}, \rightsquigarrow)$ forms a PARS. Reduction can happen at different places in a net, however the diamond property allows us to deal with this seamlessly.

Proposition 18 (Program Nets are Diamond)**.** *The PARS* $(\mathcal{N}, \rightsquigarrow)$ *satisfies the diamond property.* ☐

The proof relies on commutativity of the memory operations. Due to Th. 10, program net reduction enjoys all the good properties we have studied in Sec. 2:

Corollary 19. *The relation* $\rightsquigarrow$ *satisfies Confluence, Uniformity and Uniqueness of Normal Forms (see Th. 10).* ☐

The following two results can be obtained as adaptations of similar ones from [28].

Theorem 20 (Deadlock-Freeness of Net Reduction)**.** *Let* $\mathbf{R} = [(R, \mathtt{ind}_R, \mathbf{m})]$ *be a program net such that no* $\perp$, ? *or* ! *appears in the conclusions of the net R. If R contains cuts, a reduction step is always possible.* ☐

Corollary 21 (Cut Elimination)**.** *With the same hypothesis as above, if* $\mathbf{R} \not\rightsquigarrow$, *(*i.e. *no further reduction is possible) then* $\mathbf{R}$ *is cut free.* ☐

Example. The net in LHS of Fig. 5 can be embedded into a program net with a quantum memory of empty support: $(0, 0, 0, \ldots) \equiv$ "$|000\ldots\rangle$". It reduces according to Fig. 5, with the same memory. The next step requires a $\rightsquigarrow_{\mathrm{link}}$-rewrite step to attach a fresh address—say, 0—to the one node at surface. The H-sync node then rewrites with a $\rightsquigarrow_{\mathrm{update}}$-step, and we get the program net (A) in Fig. 7 with the "update" action applied to the memory: the memory corresponds to $\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle) \otimes |00\ldots\rangle$. From there, a choice reduction is in order: it uses the "test" action of the memory structure, which, according to Sec. 3.3.3 corresponds to the measurement of the qubit at address 0. This yields the probabilistic superposition of the program nets (B_1) and (B_2). As the net in (B_1) is the LHS of Fig. 5, it reduces to (C_1) (dashed arrow (a)), similar to (A) modulo the fact that the address 0 was not fresh: the $\rightsquigarrow_{\text{link}}$-rewrite step cannot yield 0: here we choose 1. Note that we could have chosen any other non-zero number as the address. The program net (B_2) rewrites to (C_2) (dashed arrow (b)): the weakening node erases the Y-box, and a fresh variable is allocated. In this case, the address 0 is indeed fresh and can be picked.

5. A Memory-Based Abstract Machine

In this section we introduce a class of memory-based token machines, called the MSIAM (*Memory-based Synchronous Interaction Abstract Machine*). The base rewrite system on which the MSIAM is built, is a variation[2] of the SIAM multi-token machine from [28], which we recall in Sec. 5.1. The specificity of the SIAM is to allow not only parallel threads, but also *interaction* among them, *i.e. synchronization*. Synchronization happens in particular at the sync nodes (unsurprisingly, as these are nodes introduced with this purpose), but also on the additive boxes (the $\perp$-box). The transitions at the $\perp$-box model *choice*: as we see below, when the flow of computation reaches the $\perp$-box (*i.e.* the tokens reach the auxiliary doors), it continues on one of the two sub-components, depending on the tokens which are positioned at the principal door.

The original contribution of this section is contained in Sections 5.2 through 5.4, where we use our parametric construction to define the MSIAM $\mathcal{M}_{\mathbf{R}}$ for $\mathbf{R}$ as a PARS consisting of a set of *states* $\mathcal{S}$, and a *transition relation* $\rightarrow\ \subset \mathcal{S} \times DST(\mathcal{S})$, and establish its main properties, in particular Deadlock-Freeness (Th. 25), Invariance (Th. 26) and Adequacy (Th. 27).

5.1 SIAM

Let R be a net. The SIAM for R is given by a set of states and a transition relation on states. Most of the definitions are standard.

Exponential signatures σ and *stacks* s are defined by
$\sigma ::= * \mid l(\sigma) \mid r(\sigma) \mid \lceil \sigma, \sigma \rceil \mid y(\sigma, \sigma)$
$s ::= \epsilon \mid l.s \mid r.s \mid \sigma.s \mid \delta$
where ϵ is the empty stack and . denotes concatenation. Two kinds of stacks are defined: (1.) the *formula stack* and (2.) the *box* stack. The latter is the standard GoI way to keep track of the different copies of a box. The former describes the formula path of either an occurrence α of a unit, or an occurrence $\Diamond$ of a modality, in a formula A. Formally, s is a *formula stack on* A if either $s = \delta$ or $s[A] = \alpha$ (resp. $s[A] = \Diamond$), with $s[A]$ defined as follows: $\epsilon[\alpha] = \alpha$, $\sigma.\delta[\Diamond B] = \Diamond$, $\sigma.t[\Diamond B] = t[B]$ whenever $t \neq \delta$, $l.t[B \Box C] = t[B]$ and $r.t[B \Box C] = t[C]$ (where $\Box$ is either $\otimes$ or $\parr$). We say that s *indicates* the occurrence α (resp. $\Diamond$).

Example 22. Given the formula $A = !(\perp \otimes !1)$, the stack $*.\delta$ indicates the *leftmost* occurrence of !, the stack $*.r.*.\delta$ indicates the *rightmost* occurrence of !, and $*.l[A] = \perp$.

Positions. Given a net R, the set of its *positions* $\mathtt{POS}_R$ contains all the triples (e, s, t), where e is an edge of R, s is a formula stack on the type A of e, and t (the *box stack*) is a stack of n exponential signatures, where n is the depth of e in R. We use the metavariables $\mathbf{s}$ and $\mathbf{p}$ to indicate positions. For each position $\mathbf{p} = (e, s, t)$, we define its *direction* $\mathrm{dir}(\mathbf{p})$ to be *upwards* ($\uparrow$) if s indicates an occurrence of ! or $\perp$, to be *downwards* ($\downarrow$) if s indicates an occurrence of ? or 1, to be *stable* ($\leftrightarrow$) if $s = \delta$ or if the edge e is the conclusion of a bot node. The following subsets of $\mathtt{POS}_R$ play a role in the definition of the machine:

- the set $\mathtt{INIT}_R$ of *initial positions* $\mathbf{p} = (e, s, \epsilon)$, with e conclusion of R, and $\mathrm{dir}(\mathbf{p})$ is $\uparrow$;
- the set $\mathtt{FIN}_R$ of *final positions* $\mathbf{p} = (e, s, \epsilon)$, with e conclusion of R, and $\mathrm{dir}(\mathbf{p})$ is $\downarrow$;
- the set $\mathtt{ONES}_R$ of positions (e, ϵ, t), e conclusion of a one node;
- the set $\mathtt{DER}_R$ of positions $(e, *.\delta, t)$, e conclusion of a ?d node;
- the set $\mathtt{STABLE}_R$ of the positions $\mathbf{p}$ for which $\mathrm{dir}(\mathbf{p}) = \leftrightarrow$;
- the set of starting positions $\mathtt{START}_R = \mathtt{INIT}_R \cup \mathtt{ONES}_R \cup \mathtt{DER}_R$.

SIAM ***States.*** A *state* (T, orig) of $\mathcal{M}_R$ is a set of positions $T \subseteq \mathtt{POS}_R$ equipped with an injective map $\mathrm{orig} : T \rightarrow \mathtt{START}_R$. Intuitively, T describes the current positions of the tokens, and orig keeps track of where each such token started its path.

A state is *initial* if $T \subseteq \mathtt{INIT}_R$ and orig is the identity. We indicate the (unique) initial state of $\mathcal{M}_R$ by I_R. A state T is *final* if all positions in T belong to either $\mathtt{FIN}_R$ or $\mathtt{STABLE}_R$.

With a slight abuse of notation, we will denote the state (T, orig) also by T. Given a state T of $\mathcal{M}_R$, we say that *there is a token in* $\mathbf{p}$ if $\mathbf{p} \in T$. We use expressions such as "a token moves", "crosses a node", in the intuitive way.

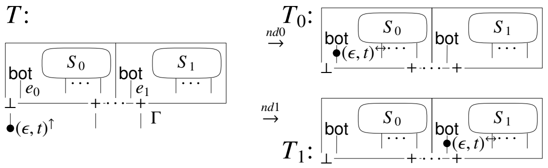

Figure 8. SIAM Non-Deterministic Transition Rules.

SIAM ***Transitions.*** The transition rules of the SIAM are described in Fig. 8 and Fig. 9. Rules (i)-(iv) require synchronization among different tokens; this is expressed by specific multi-token conditions which we discuss in the next paragraph. First, we explain the graphical conventions and give an overview of the rules.

The position $\mathbf{p} = (e, s, t)$ is represented graphically by marking the edge e with a bullet $\bullet$, and writing the stacks (s, t). A transition $T \rightarrow U$ is given by depicting only the positions in which T and U differ. It is intended that all positions of T which do not explicitly appear in the picture also belong to U. To save space, in Fig. 9 the transition arrows are annotated with a *direction*; this means that the rule applies (only) to positions which have that direction. When useful, the direction of a position is directly annotated with $^{\downarrow}$, $^{\uparrow}$ or $^{\leftrightarrow}$. Note that no transition is defined for stable positions. For boxes, whenever a token is on a conclusion of a box, it can move into that box (graphically, the token "crosses" the border of the box) and it is modified as if it were crossing a node. For exponential boxes, Fig. 9 depicts only the border of the box. We do not explicitly give the function orig, which is immediate to reconstruct when keeping in mind that it is a pointer to the origin of the token.

We briefly discuss the most interesting transitions (we refer to [28] for a broader discussion). *Fixpoints:* the recursive behavior of Y-boxes is captured by the exponential signature in the form $y(\cdot, \cdot)$, and the associated transitions. *Duplication:* the key is rule (iv), which generates a token on the conclusion of a ?d node; this token will then travel the net, possibly crossing a number of contractions, until it finds its exponential box; intuitively, each such token corresponds to *a copy of a box. One:* the behavior of the token generated by rule (iii) on the conclusion of a one node is similar to that of a dereliction token; the one token searches for its $\perp$-box. *Stable tokens:* when the token from an instance of a ?d or of a one node "has found its box", *i.e.* it reaches the principal door of a box, the token become *stable* ($\leftrightarrow$). A stable token is akin to a marker sitting on the principal door

[2] In this paper we make this transition non-deterministic; otherwise there is no major difference.

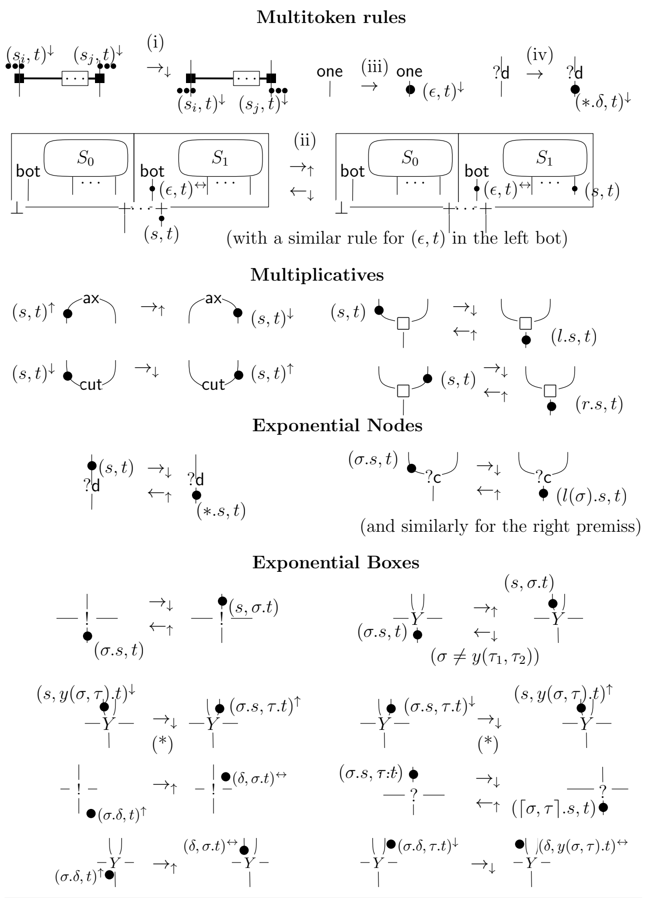
Multitoken rules
(with a similar rule for (ϵ, t) in the left bot)
Multiplicatives
Exponential Nodes
(and similarly for the right premiss)
Exponential Boxes

Figure 9. SIAM: Deterministic Transition Rules.

of its box, keeping track of the box copies and of the choice made in each specific copy.

Multi-token Conditions. The rules marked by (i), (ii), (iii), and (iv) in Fig. 9 require the tokens to interact, which is formalized by multi-token conditions. Such conditions allow, in particular, to capture choice and synchronization. Below we give an intuitive presentation; we refer to [28, 29] for the formal details.

Synchronization, rule (i). To cross a sync node l, all the positions on the premises of l (for the same box stack t) must be filled; intuitively, having the same t, means that the positions all belong to the same copy of l. Only when all the tokens have reached l, they can cross it; they do so simultaneously.

Choice, rule (ii). Any token arriving at a $\perp$-box on an auxiliary door must wait for *a token on the principal door to have made a choice* for either of the two contents, S_0 or S_1: a token (e, s, t) on the conclusions Γ of the $\perp$-box will move to S_0 (resp. S_1) *only if* the principal door of S_0 (resp. S_1) has a token with the same t.

The rules marked by *(iii) and (iv)* also carry a multi-token condition, but in a more subtle way: a token is enabled to start its journey on a one or ?d node only when its box has been opened; this reflects in the SIAM the constraint of *surface reduction* of nets.

5.2 MSIAM

Similarly to what we have done for nets, we enrich the machine with a memory, and use the SIAM and the operations on the memory to define a PARS.

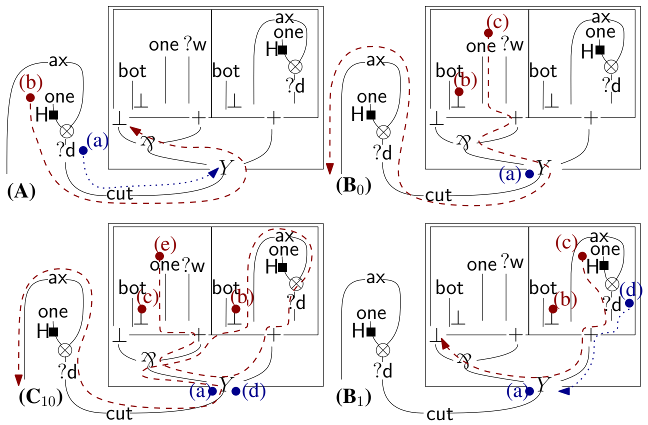

Figure 10. MSIAM run of Fig. 5.

MSIAM ***States.*** Given a memory structure $\mathrm{Mem} = (\mathrm{Mem}, I, \mathcal{L})$ and a raw program net $(R, \mathtt{ind}_R, \mathbf{m}_R)$ on Mem, a *raw state* of the MSIAM $\mathcal{M}_{\mathbf{R}}$ is a tuple $(T, \mathtt{ind}_{\mathbf{T}}, \mathbf{m}_T)$ where

- T is a state of $\mathcal{M}_R$,
- $\mathtt{ind}_T : \mathtt{START}_R \to I$ is a partial injective map,
- $\mathbf{m}_T \in \mathrm{Mem}$.

States are defined as the equivalence class $\mathbf{T} = [(T, \mathtt{ind}_T, \mathbf{m}_T)]$ of row states *over permutations*, with the action of $\mathrm{Perm(I)}$ on tuples being the natural one.

MSIAM ***Transitions.*** Let $\mathbf{R}$ be a program net, and $\mathbf{T}$ be a state $[(T, \mathtt{ind}_T, \mathbf{m}_T)]$ of $\mathcal{M}_{\mathbf{R}}$. We define the transition $\mathbf{T} \to \mu \in \mathcal{S} \times DST(\mathcal{S})$. As we did for program nets, we first give the definition on raw states. The definition depends on the SIAM transitions for T. Let us consider the possible cases.

1. *Link.* Assume $T \overset{(iii)}{\to} U$ (Fig. 9), and let n be the one node, x its conclusion, and $\mathbf{p}$ the new token in U. We set
$$(T, \mathtt{ind}_T, \mathbf{m}) \to_{\mathrm{link(n,i)}} (U, \mathtt{ind}_T \cup \{\mathrm{orig}(\mathbf{p}) \mapsto i\}, \mathbf{m})$$
where we choose $i = \mathtt{ind}_R(x)$ if the one node is active, and otherwise an address i which is fresh for both $\mathtt{ind}_T$ and $(\mathbf{m})$.
2. *Update.* Assume $T \overset{(i)}{\to} U$ (Fig. 9), l is the name associated to the sync node, and $\vec{i}$ are the addresses which are associated to its premises (by composing orig and ind), then
$$(T, \mathtt{ind}_T, \mathbf{m}) \to_{\mathrm{update(s)}} \{(U, \mathtt{ind}_T, \mathrm{update}(l, \vec{i}, \mathbf{m})^1\}.$$
3. *Test.* Assume $T \overset{nd0}{\to} T_0$ and $T \overset{nd1}{\to} T_1$ (Fig. 8, non-deterministic transition). If $\mathbf{p} \in T$ is the token appearing in the redex (Fig. 8), and i the addresses that $\mathtt{ind}_T$ associates to $\mathrm{orig}(\mathbf{p})$, then
$$(T, \mathtt{ind}_{\mathbf{T}}, \mathbf{m}) \to_{\mathrm{test(i)}}$$
$$\mathrm{test}(i, \mathbf{m})[\mathrm{false} := (T_0, \mathtt{ind}_T), \mathrm{true} := (T_1, \mathtt{ind}_T)].$$
4. In all the other cases: if $T \to U$ then $(T, \mathtt{ind}_T, \mathbf{m}) \to \{(U, \mathtt{ind}_T, \mathbf{m})^1\}$.

Let $\mathbf{R} = [(R, \mathtt{ind}_R, \mathbf{m}_R)]$. The *initial state* of $\mathcal{M}_{\mathbf{R}}$ is $\mathbf{I}_{\mathbf{R}} = [(I_R, \mathtt{ind}_{I_R}, \mathbf{m}_R)]$, where $\mathtt{ind}_{I_R}$ is only defined on the initial positions: if $\mathbf{p} \in \mathtt{INIT}_R$, and x is the occurrence of $\perp$ corresponding to $\mathbf{p}$, then $\mathtt{ind}_{I_R}(\mathbf{p}) = \mathtt{ind}_R(x)$. A state $[(T, \mathtt{ind}_T, \mathbf{m}_T)]$ is *final* if T is final.

In the next sections, we study the properties of the machine, and show that the MSIAM is a computational model for $\mathcal{N}$.

Example. We informally develop in Fig. 10 an execution of the MSIAM for the LHS net of Fig. 5. In the first panel (A) tokens (a) and (b) are generated. Token (a) reaches the principal door of the Y-box, which corresponds to *opening* a first copy. Token (b) enters the Y-box and hits the $\perp$-box. The test action of the memory triggers a probabilistic distribution of states where the left and the right components of the $\bot$-box are opened: the corresponding sequences of operations are Panels (B_0) and (B_1) for the left and right sides.

In Panel (B_0): the left-side of the $\bot$-box is opened and its one-node emits the token (c) that eventually reaches the conclusion of the net. In Panel (B_1): the right-side of the $\bot$-box is opened and tokens (c) and (d) are emitted. Token (d) opens a new copy of the Y-box, while token (c) hits the $\bot$-box of this second copy. The test action of the memory again spawns a probabilistic distribution.

We focus on panel (C_{10}) on the case of the opening of the left-side of the $\bot$-box: there, a new token (e) is generated. It will exit the second copy of the Y-box, go through the first copy and exit to the conclusion of the net.

5.3 MSIAM Properties, and Deadlock-Freeness

Intuitively, a *run* of the machine $\mathcal{M}_{\mathbf{R}}$ is the result of composing transitions of $\mathcal{M}_{\mathbf{R}}$, starting from the initial state $\mathbf{I}_{\mathbf{R}}$ (composition being transitive composition). We are not interested in the actual order in which the transitions are performed in the various components of a distribution of states. Instead, we are interested in knowing which distributions of states are reached from the initial state. This notion is captured well by the relation $\looparrowright$ (see Sec. 2.1). We will say that *a run of the machine $\mathcal{M}_{\mathbf{R}}$ reaches $\mu \in DST(\mathcal{S}_{\mathbf{R}})$ if $\mathbf{I}_{\mathbf{R}} \looparrowright \mu$.*

An analysis similar to the one done for program nets gives the next lemma (Lem. 23) and therefore Prop. 24:

Lemma 23 (Diamond)**.** *The relation $\to$ satisfies the diamond property.* □

Proposition 24 (Confluence, Uniqueness of Normal Forms, Uniformity)**.** *The relation $\to$ satisfies confluence, uniformity, and uniqueness of normal forms.* □

By the results we have studied in Sec. 2, we thus conclude that all runs of $\mathcal{M}_{\mathbf{R}}$ have the same behavior with respect to the degree of termination, *i.e.* if $\mathbf{I}_{\mathbf{R}}$ p-normalizes following a certain sequence of reductions, it will do so whatever sequence of reductions we pick.

Deadlocks. A terminal state $\mathbf{T} \not\to$ of $\mathcal{M}_{\mathbf{R}}$ can be final or not. A non-final terminal state is called a *deadlocked* state. Because of the inter-dependencies among tokens given by the multi-token conditions, a multi-token machine is likely to have deadlocks. We are however able to guarantee that any MSIAM machine is deadlock-free, whatever is the choice for the memory structure.

Theorem 25 (Deadlock-Freeness of the MSIAM)**.** *Let $\mathbf{R}$ be a program net of conclusion 1; if $\mathbf{I}_{\mathbf{R}} \looparrowright \mu$ and $\mathbf{T} \in SUPP(\mu)$ is terminal, then $\mathbf{T}$ is a* final *state.* □

5.4 Invariance and Adequacy

The machine $\mathcal{M}_{\mathbf{R}}$ gives a computational semantics to $\mathbf{R}$. The semantics is invariant under reduction (Th. 26); the adequacy result (Th. 27) relates convergence of the machine and convergence of the nets. We define the convergence of the machine as the convergence of its initial state:

$$\mathcal{M}_{\mathbf{R}} \Downarrow_p \text{ if } \mathbf{I}_{\mathbf{R}} \textit{ converges with probability } p \ (\textit{i.e. } \mathbf{I}_{\mathbf{R}} \Downarrow_p).$$

Theorem 26 (Invariance)**.** *Let $\mathbf{R}$ be a program net of conclusion 1. Assume $\mathbf{R} \rightsquigarrow \sum_i p_i \cdot \{\mathbf{R}_i\}$. Then we have that $\mathcal{M}_{\mathbf{R}} \Downarrow_q$ if and only if $\mathcal{M}_{\mathbf{R}_i} \Downarrow_{q_i}$ with $\sum_i (p_i \cdot q_i) = q$.* □

Theorem 27 (Adequacy)**.** *Let $\mathbf{R}$ be a program net of conclusion 1. Then, $\mathcal{M}_{\mathbf{R}} \Downarrow_p$ if and only if $\mathbf{R} \Downarrow_p$.* □

The proofs of invariance and adequacy, as well as that of deadlock-freeness, all are based on the *diamond property* of the machine (Lem. 23).

6. A PCF-style Language with Memory Structure

We introduce a PCF-style language which is equipped with a memory structure, and is therefore parametric on it. The base type will correspond to elements stored in the memory, and the base operations to the operations of the memory structure.

6.1 Syntax and Typing Judgments

The language $\mathsf{PCF}^{\mathsf{LL}}$ which we propose is based on Linear Logic, and is parameterized by a choice of a memory structure Mem.

The *terms* (M, N, P) and *types* (A, B) are defined as follows:

$$\begin{aligned} M,N,P ::=\ & x \mid \lambda x.M \mid MN \mid \mathtt{let}\,\langle x,y\rangle = M\,\mathtt{in}\,N \mid \langle M,N\rangle \mid \\ & \mathtt{letrec}\,f\,x = M\,\mathtt{in}\,N \mid \\ & \mathtt{new} \mid \mathtt{c} \mid \mathtt{if}\,P\,\mathtt{then}\,M\,\mathtt{else}\,N, \\ A,B \quad ::=\ & \alpha \mid A \multimap B \mid A \otimes B \mid !A \end{aligned}$$

where $\mathtt{c}$ ranges over the set of memory operations $\mathcal{L}$. A typing context Δ is a (finite) set of typed variables $\{x_1 : A_1, \ldots, x_n : A_n\}$, and a typing judgment is written as $\Delta \vdash M : A$. An empty typing context is denoted by "$\cdot$". We say that a type is *linear* if it is not of the form $!A$. We denote by $!\Delta$ a typing context with only non-linearly typed variables. A typing judgment is *valid* if it can be derived from the set of typing rules presented in Fig. 11. We require M and N to have empty context in the typing rule of $\mathtt{if}\,P\,\mathtt{then}\,M\,\mathtt{else}\,N$. The requirement does not reduce expressivity as typing contexts can always be lambda-abstracted. The typing rules make use of a notion of *values*, defined as follows: $U, V ::= x \mid \lambda x.M \mid \langle U, V\rangle \mid \mathtt{c}$.

6.2 Operational Semantics

The operational semantics for $\mathsf{PCF}^{\mathsf{LL}}$ is similar to the one of [13], and is inherently call-by-value. Indeed, being based on Linear Logic, the language only allows the duplication of "!"-boxes, that is, normal forms of "!"-type: these are the values. The operational semantics is in the form of a PARS, written $\to$.

The PARS is defined using a notion of *reduction context* $C[-]$, defined by the grammar

$$\begin{aligned} C[-] \ ::=\ & [-] \mid C[-]N \mid VC[-] \mid \langle C[-], N\rangle \mid \langle V, C[-]\rangle \\ & \mid \mathtt{let}\,\langle x,y\rangle = C[-]\,\mathtt{in}\,N \mid \mathtt{if}\,C[-]\,\mathtt{then}\,M\,\mathtt{else}\,N, \end{aligned}$$

and a notion of abstract machine: the $\mathsf{PCF}_{\mathsf{AM}}$. A *raw* $\mathsf{PCF}_{\mathsf{AM}}$ *closure* is a tuple $(M, \mathtt{ind}_M, \mathbf{m})$ where M is a term, $\mathtt{ind}_M$ is an injective map from the set of free variables of M to I, and $\mathbf{m} \in \mathrm{Mem}$. $\mathsf{PCF}_{\mathsf{AM}}$ *closures* are defined as equivalence classes of raw $\mathsf{PCF}_{\mathsf{AM}}$ closures over permutations of addresses.

The rewrite system is defined in Fig. 12. First, the creation of a new base type element ($\to_{\mathrm{link}}$) is memory allocation: x is fresh (and not bound) in C and i is a new address neither in the image of $\mathtt{ind}$ nor in the support of $\mathbf{m}$. Then, the operation $\mathtt{c}$ reduces through $\to_{\mathrm{update}(\mathtt{c})}$ using the update of the memory when $\mathrm{arity}(\mathtt{c}) = n$ and $\mathtt{ind}(x_k) = i_k$. Then, the if-then-else reduces through $\to_{\mathrm{test}(i)}$ using the test operation where $\mathtt{ind}(x) = i$. Note how we remove x from the domain of $\mathtt{ind}$. Finally we have the three rules that do not involve probabilities: Note how the mapping $\mathtt{ind}$ can be kept the same: the set of free variables is unchanged.

Let $\mathbf{M} = [(M, \mathtt{ind}, \mathbf{m})]$ be a $\mathsf{PCF}_{\mathsf{AM}}$ closure. We define the judgment $x_1 : A_1, \ldots, x_m : A_m \vdash \mathbf{M} : B$ if none of the x_i's belongs to $\mathrm{Dom}(\mathtt{ind})$, $y_1 : \alpha, \ldots, y_k : \alpha, x_1 : A_1, \ldots, x_m : A_m \vdash M : B$, and $\{y_1, \ldots, y_k\} = \mathrm{Dom}(\mathtt{ind})$.

6.3 Modeling $\mathsf{PCF}^{\mathsf{LL}}$ with Nets

We now encode $\mathsf{PCF}^{\mathsf{LL}}$ typing judgments and typed $\mathsf{PCF}_{\mathsf{AM}}$ closures into program nets. As the type system is built on top of Linear Logic, the translation $(-)^\dagger$ is rather straightforward, modulo one subtlety: it is parameterized by a memory structure $\mathbf{m}$ and a partial function $\mathtt{ind}$ mapping term variables to addresses in I.

$$\frac{}{!\Delta \vdash \texttt{new} : \alpha} \quad \frac{}{!\Delta, x : !(A \multimap B) \vdash x : A \multimap B} \quad \frac{A \text{ linear}}{!\Delta, x : A \vdash x : A} \quad \frac{!\Delta \vdash V : A \multimap B \quad V \text{ value}}{!\Delta \vdash V : !(A \multimap B)} \quad \frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x.M : A \multimap B}$$

$$\frac{!\Delta, \Gamma_1 \vdash M : A \multimap B \quad !\Delta, \Gamma_2 \vdash N : A}{!\Delta, \Gamma_1, \Gamma_2 \vdash MN : B} \quad \frac{!\Delta, \Gamma_1 \vdash M : A \otimes B \quad !\Delta, \Gamma_2, x : A, y : B \vdash N : C}{!\Delta, \Gamma_1, \Gamma_2 \vdash \texttt{let}\, \langle x, y \rangle = M \,\texttt{in}\, N : C} \quad \frac{!\Delta, \Gamma_1 \vdash M : A \quad !\Delta, \Gamma_2 \vdash N : B}{!\Delta, \Gamma_1, \Gamma_2 \vdash \langle M, N \rangle : A \otimes B}$$

$$\frac{\Delta \vdash P : \alpha \quad \cdot \vdash M : A \quad \cdot \vdash N : A}{\Delta \vdash \texttt{if}\, P \,\texttt{then}\, M \,\texttt{else}\, N : A} \quad \frac{\text{arity}(\texttt{c}) = n}{!\Delta \vdash \texttt{c} : \alpha^{\otimes n} \multimap \alpha^{\otimes n}} \quad \frac{!\Delta, f : !(A \multimap B), x : A \vdash M : B \quad !\Delta, \Gamma, f : !(A \multimap B) \vdash N : C}{!\Delta, \Gamma \vdash \texttt{letrec}\, f\, x = M \,\texttt{in}\, N : C}$$

Figure 11. Typing Rules.

$$(C[\texttt{new}], \texttt{ind}, \mathbf{m}) \to_{\text{link}} (C[x], \texttt{ind} \cup \{x \mapsto i\}, \mathbf{m}) \quad (C[\texttt{c}\, \langle x_1, \ldots, x_n \rangle], \texttt{ind}, \mathbf{m}) \to_{\text{update}(\texttt{c})} (C[\langle \vec{x} \rangle], \texttt{ind}, \text{update}(\vec{i}, \texttt{c}, \mathbf{m}))$$
$$(C[\texttt{if}\, x \,\texttt{then}\, M_{\texttt{t}} \,\texttt{else}\, M_{\texttt{f}}], \texttt{ind}, \mathbf{m}) \to_{\text{test}(i)} \text{test}(i, \mathbf{m})[\text{true} := (M_{\texttt{t}}, \texttt{ind} \setminus \{x \mapsto i\}), \text{false} := (M_{\texttt{f}}, \texttt{ind} \setminus \{x \mapsto i\})]$$
$$(C[(\lambda x.M)U], \texttt{ind}, \mathbf{m}) \to (C[M\{x := U\}], \texttt{ind}, \mathbf{m}) \quad (C[\texttt{let}\, \langle x, y \rangle = \langle U, V \rangle \,\texttt{in}\, M], \texttt{ind}, \mathbf{m}) \to (C[N[x := U, y := V]], \texttt{ind}, \mathbf{m})$$
$$(C[\texttt{letrec}\, f\, x = M \,\texttt{in}\, N], \texttt{ind}, \mathbf{m}) \to (C[N\{f := \lambda x.\texttt{letrec}\, f\, x = M \,\texttt{in}\, M\}], \texttt{ind}, \mathbf{m})$$

Figure 12. Rewrite System for $\mathsf{PCF}_{\mathsf{AM}}$.

The mapping $(-)^{\dagger}$ of types to formulas is defined by $\alpha^{\dagger} := \mathbf{1}$, $(A \multimap B)^{\dagger} := (A^{\dagger\perp} \parr B^{\dagger})$ and $(A \otimes B)^{\dagger} := A^{\dagger} \otimes B^{\dagger}$. Now, assume that $\{y_1, \ldots, y_n\} \cap \text{Dom}(\texttt{ind}) = \emptyset$, that Δ is a judgment whose variables are all of type α, and that $|\Delta| = \text{Dom}(\texttt{ind})$. The typing judgment $y_1 : A_1, \ldots, y_n : A_n, \Delta \vdash M : A$ is mapped through $(-)^{\dagger}_{\texttt{ind},\mathbf{m}}$ to a program net $M^{\dagger}_{\texttt{ind},\mathbf{m}} = [(R_M, \texttt{ind}_{R_M}, \mathbf{m})]$ with conclusions $(A_1^{\dagger})^{\perp}, \ldots (A_n^{\dagger})^{\perp}, (B^{\dagger})$ and memory state $\mathbf{m}$ (note how the variables in Δ do not appear as conclusions).

6.3.1 Adequacy

As in Sec. 2 and 5.4, given a $\mathsf{PCF}_{\mathsf{AM}}$ closure $\mathbf{M}$ we write $\mathbf{M} \Downarrow_p$ ($\mathbf{M}$ converges to p) if $p = \sup_{\mathbf{M} \rightrightarrows^* \mu} \mathcal{T}(\mu)$. The adequacy theorem then relates convergence of programs and convergence of nets.

Theorem 28. *Let $\vdash \mathbf{M} : \alpha$, then $\mathbf{M} \Downarrow_p$ if and only if $\mathbf{M}^{\dagger} \Downarrow_p$.* □

7. Results and Discussion

As we anticipated in Sec. 1.3, we have proved—*parametrically* on the *memory*—that the MSIAM is an adequate model of program nets reduction (Th. 27), and program nets are expressive enough to adequately represent the behavior of the $\mathsf{PCF}^{\mathsf{LL}}$ abstract machine (Th. 28). What does this mean? As soon as we choose a concrete instance of memory structure, we have a language and an adequacy result for it. This is in particular the case for all instances of memory which are outlined in Sec. 3.3. To make this explicit, let $\mathcal{I}$, $\mathcal{P}$ and $\mathcal{Q}$ be respectively a deterministic, probabilistic and quantum memory. We denote by $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{I})$, $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{P})$ and $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{Q})$, respectively, the language which is obtained by choosing that memory. Observe in particular that the choice of $\mathcal{P}$ or $\mathcal{Q}$, respectively specializes our adequacy result into a semantics for a probabilistic PCF in the style of [33], and a semantics for a quantum PCF, in the style of [11, 13].

7.1 The Quantum Lambda Calculus

Let us now focus on the quantum case, and analyze in some depth our result. We have a quantum lambda-calculus, namely $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{Q})$, together with an adequate multi-token semantics. How does our calculus relate with the ones in the literature?

We first observe that the *syntax* of $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{Q})$ is very close to the language of [13] (we only omit lists and coproducts). The *operational semantics* is also the same, as one can easily see. Indeed, the abstract machine in [13] consists of a triple (Q, L, M) where M is a lambda-term and where Q and L are as presented in Sec. 3.3.3. As we discussed there, for Q and L one can use either the canonical presentation of [13], or the memory structure $\mathcal{Q}$.

7.1.1 Discussion on the Quantum Model

It is now time to go back to the programs in our motivating examples, Examples 2 and 3. Both programs are valid terms in $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{Q})$; we have already informally developed Example 3 within our model.

We claimed in the Introduction that Example 2 cannot be represented in the GoI model described in [21]: the reason is that the model does not support entangled qubits in the type $\alpha \otimes \alpha$ (using our notation), a tensor product is always separable. To handle entangled states, [21] uses non-splittable, crafted types: this is why the simple term in Example 2 is forbidden. In the MSIAM, entangled states pose no problem, as the memory is disconnected from the types.

The term of Example 3, valid in $\mathsf{PCF}^{\mathsf{LL}}(\mathcal{Q})$, is mapped through $(-)^{\dagger}$ to the net of Fig. 5: Th. 28 and 27 state that the corresponding MSIAM presented in Fig. 10 is adequate. Note that Example 3 was presented in the context of quantum computation. It is however possible (and the behavior is going to be the same as the one already described) to use the probabilistic memory sketched in Sec. 3.3.2. In this case, the H-sync node would be changed for the coin-sync node.

7.1.2 Qubits, Duplication and Erasing

It is worth to pinpoint the technical ingredients which allow for the coexistence of quantum bits with duplication and erasing. In the language, the reason is that, similarly to [13], $\mathsf{PCF}^{\mathsf{LL}}$ allows only lambda-abstractions (or tuples thereof) to be duplicated. In the case of the nets (and therefore of the MSIAM), the key ingredient is *surface reduction* (Sec. 4.2): the allocation of a quantum bit is captured by the *link* rule which associates a one node to the memory. Since a one node linked to the memory *cannot* lie inside a box, it will *never be copied nor erased*. Indeed, the ways in which the language and the model deal with quantum bits, actually match.

7.2 Conclusion

In this paper, we have introduced a parallel, multi-token Geometry of Interaction capturing the choice effects with a parametric memory. This way, we are able to represent *classical, probabilistic and quantum effects*, and adequately model the linearly-typed language $\mathsf{PCF}^{\mathsf{LL}}$ parameterized by the same memory structure. We expect our approach to capture also non-deterministic choice in a natural way: this is ongoing work.

References

[1] K. de Leeuw, E. F. Moore, et al. Computability by probabilistic machine. In *Automata Studies*. Princeton U. Press, 1955.

[2] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 10th edition, 2011.

[3] G. L. Miller. Riemann's hypothesis and tests for primality. In *STOC*, pages 234–239, 1975.

[4] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128 – 138, 1980.

[5] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *SFCS*, pages 124–134, 1994.

[6] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik. Simulation of electronic structure hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[7] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[8] D. Kozen. Semantics of probabilistic programs. *Journal of Computer System Sciences*, 22:328–350, 1981.

[9] G. Plotkin. Probabilistic powerdomains. In *CAAP*, 1982.

[10] S. J. Gay. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.

[11] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Math. Struct. in Comp. Sc.*, 16(3):527–552, 2006.

[12] A. S. Green, P. LeFanu Lumsdaine, et al. Quipper: A scalable quantum programming language. In *PLDI*, pages 333–342, 2013.

[13] M. Pagani, P. Selinger, and B. Valiron. Applying quantitative semantics to higher-order quantum computing. In *POPL*, pages 647–658, 2014.

[14] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[15] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[16] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[17] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. *Logic Colloquium 88*, 1989.

[18] Y. Lafont. *Logiques, catégories et machines*. PhD thesis, Université Paris 7, 1988.

[19] P.-A. Melliès. Categorical semantics of linear logic. *Panoramas et Synthèses*, 12, 2009.

[20] P. Selinger. Towards a quantum programming language. *Math. Struct. in Comp. Sc.*, 14(4):527–586, 2004.

[21] I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS*, pages 237–246, 2011.

[22] Y. Delbecque. *Quantum games as quantum types*. PhD thesis, McGill University, 2009.

[23] Y. Delbecque and P. Panangaden. Game semantics for quantum stores. *Electr. Notes Theor. Comput. Sci.*, 218:153–170, 2008.

[24] U. Dal Lago and M. Zorzi. Wave-style token machines and quantum lambda calculi. In *LINEARITY*, pages 64–78, 2014.

[25] U. Dal Lago, C. Faggian, I. Hasuo, and A. Yoshimizu. The geometry of synchronization. In *CSL-LICS*, pages 35:1–35:10, 2014.

[26] V. Danos and L. Regnier. Reversible, irreversible and optimal lambda-machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.

[27] Ian Mackie. The geometry of interaction machine. In *POPL*, pages 198–208, 1995.

[28] U. Dal Lago, C. Faggian, B. Valiron, and A. Yoshimizu. Parallelism and synchronization in an infinitary context. In *LICS*, pages 559–572, 2015.

[29] U. Dal Lago, C. Faggian, B. Valiron, and A. Yoshimizu. The geometry of parallelism: Classical, probabilistic, and quantum effects (long version). Available at `https://arxiv.org/abs/1610.09629`, 2016.

[30] I. Mackie. *Applications of the Geometry of Interaction to language implementation*. Phd thesis, University of London, 1994.

[31] D. R. Ghica, A. Smith, and S. Singh. Geometry of synthesis IV. In *ICFP*, pages 221–233, 2011.

[32] S. Abramsky, E. Haghverdi, and P. J. Scott. Geometry of interaction and linear combinatory algebras. *Math. Struct. in Comp. Sc.*, 12(5): 625–665, 2002.

[33] T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *POPL*, pages 309–320, 2014.

[34] V. Danos and R. Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[35] N. Hoshino, K. Muroya, and I. Hasuo. Memoryful geometry of interaction. In *CSL-LICS*, page 52, 2014.

[36] S. Staton. Algebraic effects, linearity, and quantum programming languages. In *POPL*, pages 395–406, 2015.

[37] B. Coecke, R. Duncan, A. Kissinger, and Q. Wang. Strong complementarity and non-locality in categorical quantum mechanics. In *LICS*, pages 245–254, 2012.

[38] D. R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In *POPL*, pages 363–375, 2007.

[39] K. Muroya, N. Hoshino, and I. Hasuo. Memoryful geometry of interaction II: recursion and adequacy. In *POPL*, pages 748–760, 2016.

[40] M. Bezem and J. W. Klop. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, chapter Abstract Reduction Systems. Cambridge University Press, 2003.

[41] A. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.